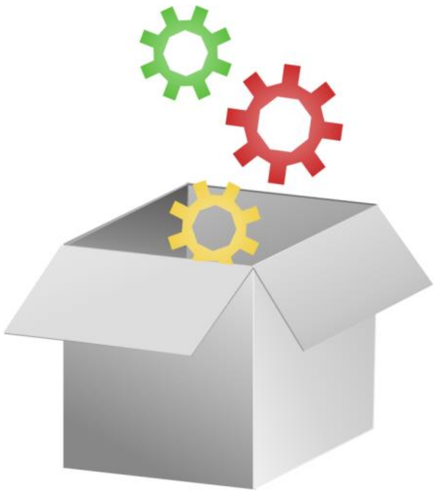
MethodSCRIPT SDK Example - Arduino



PalmSens
Compact Electrochemical Interfaces

Last revision: March 22 2019

1.1 Contents:

The arduino example “MethodScriptExample.ino” found in the “/MethodScriptExample-Arduino” folder demonstrates basic communication with the EmStat Pico through Arduino MKR ZERO using the MethodSCRIPT SDK (C libraries). The example allows the user to start measurements on the EmStat Pico from the PC connected to the Arduino through USB.

1.2 Hardware setup:

- To run this example, connect your Arduino MKRZERO "Serial1" port Rx (pin 13), Tx (pin 14) and GND to the EmStat Pico “Serial” Tx, Rx and GND respectively.
- Make sure the UART switch block SW4 on the EmStat dev board has the switches for MKR 3 and 4 turned on.
- The Arduino board should be connected normally to a PC.
- If not powering the EmStat by other means, the EmStat Pico should be connected to the PC through USB for power.

1.3 Environment setup:

- To run this example, you must include the MethodSCRIPT C libraries first.
- To do this, follow the menu "Sketch -> Include Library -> Add .ZIP/Library..." and select the MethodScriptComm folder.

1.4 How to use

- Compile and upload this sketch through the Arduino IDE.
- Next, open a serial monitor to the Arduino (you can do this from the Arduino IDE).
- You should see messages being printed containing measured data from the EmStat Pico.

1.5 Communications

In order to use the C library, MSComm, the “extern C” wrapper has to be used because Arduino uses a C++ compiler.

```
extern "C" {
  #include <MSComm.h>
  #include <MathHelpers.C>
};
```

As MSComm is the communication object with the EmStat Pico it needs some read/write functions to be passed in through the MSCommInit(in MSCommInit). However, because the C compiler doesn't understand C++ classes, the write/read functions from the Serial class are wrapped in a normal function, first as shown below.

```
int write_wrapper(char c)
{
  if(_printSent == true)
  {
    //Send all data to PC as well for debugging purposes
    Serial.write(c);
  }
  return Serial1.write(c);
}

int read_wrapper()
{
  int c = Serial1.read();
  if(_printReceived == true && c != -1) //-1 means no data
  {
    //Send all received data to PC for debugging purposes
    Serial.write(c);
  }
  return c;
}
```

A new UART instance is then created and assigned to TX (14) and RX (13) pins on the Arduino.

```
Uart Serial1(&sercom5, 14, 13, SERCOM_RX_PAD_3, UART_TX_PAD_2);
```

1.5.1 Connecting to the device

The code within the setup() function is executed only once.

Assign the Serial Data Line (SDA) and Serial Clock Line (SCL) to the pins 13 and 14 respectively.

```
//Assign SDA (serial data line) function to pin 13
pinPeripheral(13, PIO_SERCOM_ALT);
//Assign SCL (serial clock line) function to pin 14
pinPeripheral(14, PIO_SERCOM_ALT);
```

1.5.2 Sending the method script

The measurement configuration parameters can be either stored in a sd card in the Arduino and read from it or stored in a constant string. For example, in the above example, the measurement configuration parameters are stored in a constant char array as below.

```
//LSV measurement configuration parameters
const char* LSV_METHOD_SCRIPT = "e\n"
                               "var c\n"
                               "var p\n"
                               "set_pgstat_mode 3\n"
                               "set_max_bandwidth 200\n"
                               "set_cr 500u\n"
                               "set_e -500m\n"
                               "cell_on\n"
                               "wait 1\n"
                               "meas_loop_lsv p c -500m 500m 50m 100m\n"
                               "pck_start\n"
                               "pck_add p\n"
                               "pck_add c\n"
                               "pck_end\n"
                               "endloop\n"
                               "cell_off\n\n";
```

The measurement configuration parameters are then sent to the arduino.

1.5.3 Receiving response

The code to receive and parse the response from the device is written in the loop() so that it runs repeatedly and the response can be ontained from the device as and when it is available.

In order to read and parse the response from the device, the Receive Package function from the MSComm library can be used in the arduino code as below.

```
RetCode code = ReceivePackage(&_msComm, &data);
```

1.5.4 Parsing the response

Each line of response returned by the function `ReadBuf() in MSComm.c library`, can be further parsed if it is identified to be a data package. Here's a sample response (raw data) from a Linear sweep voltammetric measurement.

```
eM0000\n
Pda7F85F3Fu;ba48D503Dp,10,288\n
Pda7F9234Bu;ba4E2C324p,10,288\n
Pda806EC24u;baAE16C6Dp,10,288\n
Pda807B031u;baB360495p,10,288\n
*\n
\n
```

While parsing the response, various identifiers are used to identify the type of response packages. For example, In the above sample response package,

1. 'e' marks the beginning of a response.
2. 'M' marks the beginning of a measurement loop.
3. 'P' marks the beginning of a row of data package.
4. "*\n" marks the end of measurement.
5. "\n" marks the end of response.

The following information can be found in the data packages received from the device.

- Potential (set cell potential in V)
- Current (measured current in A)

In case of Impedance spectroscopy measurements, the following data values can be obtained from the response.

- Frequency (set frequency in Hz)
- Real part of complex Impedance (measured impedance Ohm)
- Imaginary part of complex Impedance (measured impedance in Ohm)

The following meta data values if present can also be obtained from the data packages.

- CurrentStatus (OK, underload, overload, overload warning)
- CurrentRange (the current range in use at the moment)
- Noise (Noise)

1.5.4.1 Parsing the parameter values

Each row of data package begins with the header 'P'. The parameters from the data package line can be then split in to tokens based on the delimiter ';'. Each of the parameters separated from the package line can be then parsed to get the actual values of the parameters.

The initial two characters of every parameter identifies the parameter.

```
strncpy(paramIdentifier, param, 2); //Splits the parameter identifier string
```

The potential readings are identified by the string "da"
The current readings are identified by the string "ba"
The frequency readings are identified by the string "dc"
The real impedance readings are identified by the string "cc"
The imaginary impedance readings are identified by the string "cd"

For example, in the sample package seen above, the parameter identifiers are
'da7F85F3Fu' - 'da' Potential reading and
'ba48D503Dp,10,288' – 'ba' current reading.

The following 8 characters hold the parameter value

The parameter value for current reading (8 characters) from the above sample package is '48D503Dp'. This value is further parsed to retrieve the actual parameter value with the respective unit prefix.

```
//Retrieves the actual parameter value
parameterValue = (float)GetParameterValue(paramValue);
```

The SI unit prefix from the package can be obtained from the parameter value at position 8

```
//Identify the SI unit prefix from the package at position 8
char charUnitPrefix = paramValue[7];
```

In the above sample package, the unit prefix for current data is 'p' which is 1e-12 A.

After obtaining the parameter identifier and the parameter values from the package, the meta data values if present can be parsed. Meta data values if present are separated by the demiliter character ','.

```
//Rest of the parameter is further parsed to get meta data values
ParseMetaDataValues(param + 10, retData);
```

1.5.4.2 Parsing the meta data values

The meta data values are separated based on the delimiter ',' and each of the values is further parsed to get the actual value.

The first character of each meta data value metaData[0] identifies the type of meta data.

'1' – status
'2' – Current range index
'4' - Noise

The status is 1 character hex bit mask. It is converted to long int. The status can be obtained as shown in the code snippet below.

```
char *ptr;
long statusBits = strtol(&metaDataStatus[1], &ptr , 16);
```

For example, in the above sample, the available meta data values for current data are,
10,288. The first meta data value is 10.

1 – meta data status – 0 indicates OK.

The meta data type current range is 2 characters long hex value. If the first bit high (0x80) it indicates a high speed mode current range.

The code below can be used to get current range bits from the package.

```
char  crBytePackage[3];
char* ptr;
//Fetches the current range bits from the package
strncpy(crBytePackage, metaDataCR+1, 2);
```

The hex value is then converted to int to get the current range string as shown below.

```
char* currentRangeStr;
int crByte = strtol(crBytePackage, &ptr, 16);
```

For example, in the above sample, the second meta data available is 288.

2 – indicates the type – current range
88 – indicates the hex value for current range index – 1mA. The first bit 8 implies that it is high speed mode current range.